Búsqueda Neuronal de Recursos Adaptativa en Sistemas Peer-to-Peer (BNAP2P)

Leonardo Corbalán, Laura Lanzarini, Armando De Giusti

III-LIDI (Instituto de Investigación en Informática LIDI)
Facultad de Informática. Universidad Nacional de La Plata
La Plata, Buenos Aires, Argentina
{corbalan, laural, degiusti}@lidi.info.unlp.edu.ar

Resumen. Las redes Peer-to-Peer (P2P) desestructuradas como Gnutella constituyen sistemas distribuidos con características deseables: autonomía, descentralización, auto-organización, dinamismo y tolerancia a fallas. En contrapartida, la busqueda eficiente de recursos suele ser un problema complejo. La Búsqueda Neuronal de Recursos ha demostrado ser una buena solución. En este artículo se extiende dicha solución proponiendo y evaluando un conjunto de métodos de adaptación que permiten al sistema seguir trabajando en forma eficiente ante los cambios tan habitulaes en las redes P2P.

Palabras Claves: Sistemas distribuidos, Redes Peer-to-Peer, Gnutella Búsqueda de Recursos, Redes Neuronales, Adaptación, Aprendizaje, LVQ.

1 Introducción

Las redes *Peer-To-Peer* (P2P) son sistemas compuestos por nodos pares —o iguales— que intercambian información actuando como servidores y clientes al mismo tiempo. En el modelo P2P puro —*Gnutella*, *Kademlia*, etc— computadoras individuales se comunican directamente con otras sin ningún tipo de control central. Se caracterizan por su autonomía, descentralización, auto-organización, dinamismo y tolerancia a fallas, pero también por presentar dificultades para proveer mecanismos eficientes de búsqueda de recursos. Para hacer frente a este problema *Gnutella* utiliza un algoritmo *Breadth-First Search* (BFS) realizando una difusión de solicitudes limitada en profundidad por un contador decreciente *Time-To-Live* (TTL) que impide el colapso de la red.

Si bien la estrategia BFS de *Gnutella* es atractiva por su simplicidad, no es una solución escalable y se han propuesto varios métodos para mejorar este algoritmo [1], *Expanding Ring*, *Random Walk*, y *Modified Random BFS* [2] son sólo algunos ejemplos. También se han propuestos soluciones basadas en heuristicas [3] [4] y otras que utilizan redes neuronales artificiales como *NeuroSearch* [5].

Las estrategias basadas en tablas de *hash* distribuidas —*Distributed Hash Tables* (DHTs)— utilizadas en sistemas P2P estructurados han logrado buenos resultados, sin embargo la transitoriedad de los nodos, tan característica en estos sistemas, causa pocos problemas a las redes desestructuradas como *Gnutella*, pero ocasiona un

importante gasto general —*overhead*— en los sistemas que utilizan DHTs [6]. Las DHTs aventajan a los sistemas tipo *Gnutella* cuando el recurso buscado es escaso, pero si existen muchas instancias del mismo —como es habitual— *Gnutella* puede hallarlo fácilmente [7]

En [8] hemos propuesto una nueva estrategia de búsqueda de recursos en redes P2P puras desestructuradas, utilizando redes neuronales artificiales —de aquí en adelante referida como Búsqueda Neuronal en Redes P2P (BNP2P)—. BNP2P confiere a cada nodo del sistema la "inteligencia" necesaria para conducir la búsqueda sólo hacia los vecinos más prometedores, disminuyendo el tráfico sobre la red y haciendo más eficiente la utilización del ancho de banda disponible.

En [9] hemos extendido BNP2P con una estrategia incremental que reduce significativamente el tráfico generado sobre la red. En esta estrategia la búsqueda se va expandiendo incrementalmente eligiendo los siguientes mejores vecinos aún no visitados —búsqueda más ancha—. En ambas soluciones el aprendizaje tradicional de las redes neuronales que asisten a los nodos en la toma de decisiones ha sido reemplazado por un adecuado intercambio de información entre vecinos antes de poner a funcionar todo el sistema. Sin embargo, aún no se ha presentado ningún análisis sobre posibles métodos de adecuación de los pesos de las redes neuronales para hacer frente a los cambios de topología y de recursos compartidos, tan frecuente en las redes P2P. Es precisamente ese aspecto el que cubre el presente artículo.

2 Búsqueda de Recursos en la Red Gnutella

Gnutella v0.4 es un protocolo sencillo, abierto y descentralizado basado en conexiones entre nodos llamados *servents* (conjunción de los términos *server-client*) [10] pues cumplen el doble rol de cliente y servidor a la vez.

Los *servents* sólo interactúan con otros *servents* conectados directamente, los que constituyen sus respectivas vecindades. Para implementar la búsqueda de recursos, Gnutella utiliza dos tipos de mensajes: *Query* y *QueryHit*. Todos los mensajes están formados por una cabecera seguida generalmente por una sección de datos. Entre otros campos, en la cabecera se encuentran el ID que se utiliza como identificador de una transacción, y el TTL que representa el tiempo de vida del mensaje en el sistema.

Para la búsqueda de recursos, *Gnutella* implementa un algoritmo *Breadth-First Search* (BFS). Un *servent* envía un mensaje *Query* de solicitud a todos sus vecinos con los que mantiene una conexión TCP directa. Éstos contestan al emisor con un mensaje *QueryHit* en caso de poseer el recurso buscado e, independientemente de ello, reenvían la solicitud a todos sus otros vecinos haciendo que el mensaje *Query* se propague hacia nodos más alejados. Los resultados positivos de la solicitud, mensajes *QueryHit*, se irán enviando al *servent* emisor recorriendo el camino inverso realizado por los mensajes de *Query* (Fig. 1).

Si un *servent* recibe el mismo *Query* más de una vez simplemente lo descarta —los mensajes son individualizados por el ID de su cabecera—. Esto evita ciclos innecesarios.

Los nodos decrementan en uno el campo TTL de los mensajes antes de propagarlos a sus vecinos. Si este valor llega a cero el mensaje es descartado. Así se establece el

alcance de cada mensaje y se evita que la red colapse debido al crecimiento exponencial de mensajes de *Query* conforme aumenta el tamaño de la red.

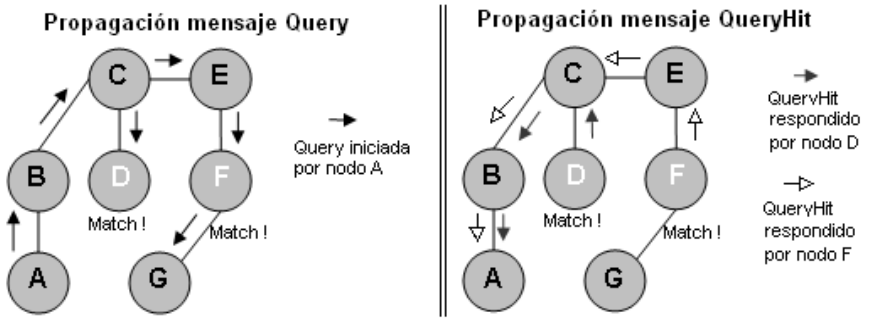

Fig. 1. Propagación de mensajes *Query* y *QueryHit* en *Gnutella*. Observe que un nodo puede recibir más de un mensaje *QueryHit* desde un mismo vecino en respuesta a su solicitud.

Gnutella no escala bien hasta los niveles que actualmente demandan las aplicaciones en *Internet*. La utilización del campo TTL en la cabecera de los mensajes impide el colapso de la red pero limita la propagación de las solicitudes al radio establecido por este valor. Los nodos más alejados, aún siendo potenciales servidores del recurso buscado, quedan fuera del alcance para todas las búsquedas iniciadas desde ese nodo en particular.

BNP2P propone un mejor criterio que la distancia para determinar cuáles serán los nodos alcanzables por una búsqueda determinada. Cada nodo tiene en cuenta las características del recurso buscado y el perfil conocido de sus vecinos para decidir por cuáles de ellos propagar una solicitud. Estas decisiones son tomadas en base a una red neuronal local con la que el nodo infiere el subconjunto más adecuado de vecinos para propagar la búsqueda.

3 Búsqueda Neuronal de Recursos en Sistemas P2P (BNP2P)

En el momento en que un nodo de la red debe propagar un mensaje *Query*, ciertos vecinos pueden resultar más prometedores que otros. BNP2P aporta a los nodos el conocimiento sobre qué clases de recursos pueden hallarse más rápidamente a partir de cada uno de sus vecinos. Esta información es utilizada para propagar los mensajes selectivamente por los caminos más adecuados haciendo un uso más eficiente del ancho de banda disponible. Así es posible utilizar valores más grandes de TTL y alcanzar nodos más alejados

En cada nodo de la red P2P se implementa una red neuronal de dos capas inspirada en las arquitecturas competitivas con el ánimo de clasificar los recursos buscados en tantas clases como vecinos posea el nodo. Sin embargo, a diferencia de la mayoría de las arquitecturas competitivas, resultan significativos los valores de activación de todas las neuronas de la última capa, y no únicamente el de la ganadora, para establecer una valoración sobre la pertenencia del recurso a cada una de las clases disponibles. Así la salida de la red establece un ranking entre los vecinos, que permite

conducir la búsqueda a través de los nodos más prometedores —aquellos que ocupan las primeras posiciones de este ranking— para ese recurso en particular. La identificación de los mejores vecinos indica que la probabilidad de hallar el recurso rápidamente, o de hallar la mayor cantidad de instancias de éste, es más alta si la búsqueda se conduce a través de ellos.

Es necesaria una representación vectorial de los recursos compartidos. La forma de construir los vectores característicos depende de las clases de recursos que se comparten en la red y la aplicación distribuida que corre sobre la misma.

El funcionamiento del sistema es el siguiente: El nodo que inicia la búsqueda de un recurso estimula su red neuronal con la representación vectorial **v** de dicho recurso. Así obtiene el ranking de nodos vecinos, un orden de preferencia para conducir la búsqueda de ese recurso en particular. Denotamos esta acción con la expresión **rank** = *rn*(**v**). La dimensión de **rank** está determinada por el número de nodos vecinos y cada una de sus componentes representa la valoración sobre la conveniencia de propagar la solicitud del recurso por el nodo correspondiente.

Dado un vector de entrada **v**, la salida de cada neurona n_j de la capa competitiva se calcula de la siguiente manera:

$$n_j(\mathbf{v}) = \frac{1}{1+\sqrt{(v_1 - w_{j_1})^2 + \ldots + (v_n - w_{jn})^2}} \qquad (\mathbf{1})$$

Siendo $\mathbf{w_j}$ el vector de peso de la neurona n_j evaluada. Así la respuesta de la neurona será mayor cuanto más cercano sea el vector **v** al vector de pesos $\mathbf{w_j}$ de la neurona.

En la figura 2 puede verse un ejemplo en el que el nodo central propaga una solicitud de recursos a un subconjunto de vecinos. Un parámetro importante del sistema será la proporción de selección (PS) que indica la cantidad de vecinos que serán seleccionados. En el ejemplo, si PS=0,5 el nodo sólo propagará el mensaje *Query* a la primera mitad del ranking de vecinos. Si PS = 1 el algoritmo de búsqueda se comporta como el BFS de *Gnutella*.

Haciendo PS < 1 se generan menos mensajes *Query* y se ahorra tráfico sobre la red. Si las decisiones locales llevadas a cabo por los nodos son acertadas, se logrará un buen promedio de hallazgos con mucho menos cantidad de mensajes.

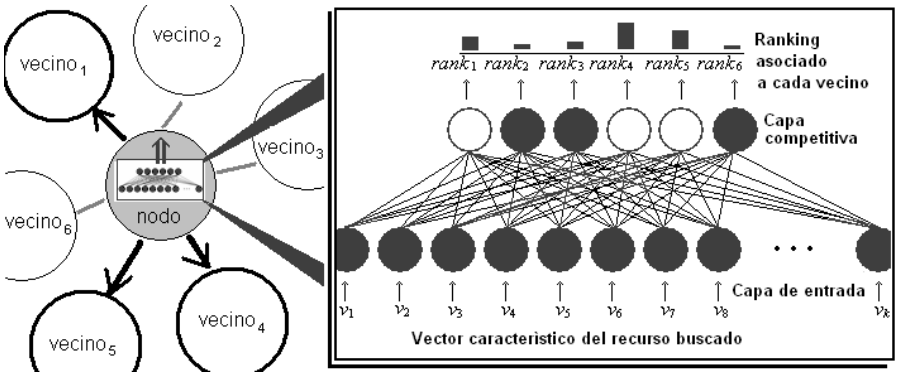

Fig. 2. Red neuronal participando en decisiones de encaminamiento de un mensaje *Query*.

La utilización de valores de TTL más grandes que en el algoritmo BFS de *Gnutella* puede ser aún más beneficiosa, alcanzando nodos más lejanos poseedores del recurso, pero manteniendo el tráfico generado en niveles aceptables. La búsqueda se hace entonces más profunda pero menos ancha. La elección del valor de TTL con el que los nodos iniciarán las solicitudes y el parámetro PS —profundidad y ancho de la búsqueda— resultan significativos en el rendimiento del sistema. En [8] se presentan varios casos estudiados con distintas asignaciones de valores a estos parámetros.

Al igual que en *Gnutella* los *QueryHits* se propagan siguiendo el camino inverso de los mensajes de *Query*. Esto permite a los nodos que propagaron la solicitud conocer si la decisión realizada para conducir la búsqueda fue exitosa o no lo ha sido. La estrategia híbrida presentada en este artículo (ver apartado 4.3) toma ventaja de esta situación.

4 BNAP2P. Estrategia Adaptativa

En BNP2P [8] no se define una estrategia completa de adaptación sino que se presenta un mecanismo basado en la propagación de información de actualización utilizando un tipo especial de mensaje llamado *NeuralInf*. Si bien se remarca la necesidad de propagaciones periódicas, no se dan precisiones sobre la frecuencia de las mismas. Tampoco se analiza el impacto sobre el tráfico generado y se presenta como una estrategia inicial para dar comienzo al funcionamiento del sistema P2P. BNAP2P completa BNP2P especificando un mecanismo adaptativo completo para mantener actualizado el conocimiento de las redes neuronales ante cambios en el ambiente.

Tradicionalmente el aprendizaje de una red neuronal competitiva se lleva a cabo por entrenamiento. Éste consiste en una fase iterativa de sucesivos ajustes de sus pesos hasta lograr la correcta clasificación de un conjunto de vectores elegidos para entrenarla. Típicamente, la capa competitiva de una red neuronal entrenada habrá establecido los vectores de pesos de cada una de sus neuronas cerca del centroide —vector promedio— del cúmulo de vectores consistente en una clase. Este hecho puede aprovecharse para acelerar el proceso de entrenamiento en caso de conocer el conjunto de todos los vectores que componen una clase. Desde el punto de vista de un nodo de la red P2P, este conocimiento lo poseen sus vecinos directos, y es precisamente este conocimiento el que se propaga en los mensajes *NeuralInf*.

4.1 Información de Actualización

Sea V el conjunto de vectores característicos de los recursos compartidos por un nodo, $V=\{\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_p\}$. El nodo calculará su propio vector centroide $\mathbf{c}$ como el promedio de todos los vectores que caracterizan sus recursos compartidos

$$\mathbf{c} = \frac{1}{p}\sum_{i=1}^{p}\mathbf{v}_i \qquad (2)$$

Sin embargo este vector **c** no será el que propague a todos sus vecinos. Ello ocasionaría un sistema de búsqueda "corto de vista" donde las redes neuronales no podrían conducir acertadamente los mensajes *Query*, salvo que el recurso se hallase en un vecino conectado al nodo en forma directa. Es necesario entonces transmitir información de nodos más alejados en los mensajes *NeuralInf*.

Un nodo *n* calcula el vector centroide de actualización en forma individualizada para cada uno de sus vecinos. Para ello utiliza la información previa recibida desde todos los otros vecinos en mensajes *NeuralInf*. Sea *s* un parámetro escalar positivo menor que 1 ($0 \leq s < 1$) que representa la significación asignada a la información proveniente de los vecinos. Sea **c** el centroide del propio nodo *n* calculado como se ha definido en la ecuación (2) y sean $\mathbf{c}_1$, $\mathbf{c}_2$, ... $\mathbf{c}_k$ los vectores centroides de actualización recibidos en mensajes *NeuralInf* por cada uno de los *k* vecinos de *n*. El cálculo del vector centroide de actualización $\mathbf{c}^i$ que se envía en un mensaje *NeuralInf* al nodo vecino *i* se calcula de la siguiente manera:

$$\mathbf{c}^i = (1-s)\,\mathbf{c} + s\,\frac{1}{k-1}\sum_{\substack{j=1 \\ j\neq i}}^{k}\mathbf{c}_j \qquad (3)$$

Así la información propagada en mensajes *NeuralInf* circula hacia nodos lejanos, aunque su intensidad vaya disminuyendo con la distancia, conforme los mensajes atraviesan la red. De esta forma un mensaje *NeuralInf* enviado por un nodo *n* no caracteriza únicamente a los recursos compartidos por *n*, sino también a todos aquellos compartidos por los nodos alcanzables desde *n*.

4.2 Propagación de la Información de Actualización

Se han evaluado distintas estrategias para propagar la información de actualización. El objetivo es generar el menor tráfico posible manteniendo aceptablemente actualizados los pesos de las redes neuronales en cada nodo.

Enfoque *Naive*. La estrategia más simple e ingenua consiste en que cada nodo, ante la evidencia de una variación en el ambiente —la baja de un nodo vecino, un cambio interno en el conjunto de recursos compartidos, etc.— reaccione inmediatamente actualizando su red neuronal y comenzando la difusión de mensajes *NeuralInf* hacia todos sus nodos vecinos. Por su parte, los nodos que reciban un mensaje *NeuralInf*, harán lo propio, actualizando sus redes neuronales y difundiendo mensajes de actualización a todos sus otros vecinos. De esta forma la información del cambio alcanzará todos los nodos de la red P2P manteniendo sus redes neuronales actualizadas. Claramente este enfoque no es escalable, los requerimientos de ancho de banda son excesivos (ver apartado 5).

Difusión limitada en profundidad (DLP). Resulta obligado limitar la propagación de los mensajes *NeuralInf* para evitar que inunden la red P2P con efectos desastrosos sobre el rendimiento general de la misma. Alcanza con añadir un campo TTL al encabezado de los mensajes *NeuralInf* para poder implementar la difusión de

mensajes de actualización con tiempo de vida. Este enfoque ha mostrado muy buenos resultados. Ello se debe a que la significación de los datos comunicados por un nodo en los mensajes *NeuralInf* va disminuyendo conforme se aleja del emisor al atravesar la red **(**ver fórmula 3) y por lo tanto resulta innecesario propagarlos hasta nodos muy distantes.

Difusión Limitada en Profundidad y Anchura (DLPA). Además de utilizar el valor de TTL para controlar el tiempo de vida de los mensajes *NeuralInf*, en este enfoque los nodos sólo propagan la información de actualización a un subconjunto de vecinos utilizando algún criterio de selección como el menos recientemente usado —LRU, por sus siglas en inglés—. El rendimiento de esta estrategia depende fuertemente del grado de acoplamiento de los nodos en la red.

Reducción de la Sensibilidad al Cambio (RSC). Una última alternativa consiste en propagar los mensajes *NeuralInf*, a cada vecino *i*, sólo si el cambio detectado, o el arribo de un mensaje *NeuralInf* desde otro vecino produce un vector centroide de actualización $\mathbf{c}^i$ cuya distancia euclidea al último vector comunicado al vecino *i* sea mayor a un cierto umbral *u*. Esto ayuda a bajar la sensibilidad a pequeños cambios en el sistema que no deberían generar tráfico en la red pues no afectan el rendimiento del sistema de búsqueda. Sin embargo, determinar el valor óptimo para este parámetro no es trivial y depende fuertemente de la topología de la red, y del conjunto de recursos compartidos por los vecinos cercanos.

4.3 Enfoque Híbrido.

Se propone espaciar en el tiempo la difusión de mensajes *NeuralInf*. Compensa el desajuste provocado por los cambios un adecuado algoritmo de aprendizaje en línea, conforme los nodos participan en decisiones de búsquedas, que no produce tráfico alguno sobre la red P2P, sino que se implementa por medio del monitoreo de mensajes *Query* enviados y mensajes *QueryHit* recibidos por el nodo.

Learning Vector Quantization (LVQ) describe una clase de algoritmos de aprendizaje supervisado. Este tipo de algoritmos necesita conocer si cada una de las respuestas de la red neuronal resultó acertada o no. En particular, en este trabajo se ha utilizado una adaptación de LVQ1 cuya regla de aprendizaje se describe a continuación.

Sea **v** el vector de entrada, **w** el vector de pesos de la neurona que gana la competición y *t* un instante de tiempo dado, la adecuación de **w** se realiza de la siguiente manera:

$$\mathbf{w}(t+1) = \begin{cases} \mathbf{w}(t) + \alpha(t)[\mathbf{v} - \mathbf{w}(t)], \text{ si } \mathbf{v} \text{ y } \mathbf{w} \text{ pertenecen a la misma clase} \\ \mathbf{w}(t) - \alpha(t)[\mathbf{v} - \mathbf{w}(t)], \text{ si } \mathbf{v} \text{ y } \mathbf{w} \text{ pertenecen a distintas clases} \end{cases} \quad (4)$$

Aquí $\alpha(t)$ representa la velocidad de aprendizaje y se recomienda comenzar con un valor pequeño que suele hacerse decrecer linealmente hasta cero, a medida que la red

va convergiendo hacia la clasificación correcta. Obsérvese que el entrenamiento consiste en reforzar los aciertos de la red neuronal, acercando el vector **w** de la neurona ganadora al vector de entrada **v**, y por el contrario, alejarlo en caso que la clasificación haya resultado errónea.

Implementación LVQ1. Debido al mecanismo de propagación de los mensajes *Query* y las respuestas *QueryHit* **(**ver fig.1), todos los nodos de la red P2P involucrados en una búsqueda conocen el resultado de la misma para cada uno de los vecinos por los cuales fue propagada. Esta información es suficiente para aplicar de forma continua un entrenamiento basado en LVQ1.

Una vez vencido el tiempo de espera a partir de la propagación de un mensaje *Query*, el nodo actualiza los pesos de todas las neuronas involucradas —aquellas con las que se identificaron los vecinos para propagar los mensajes *Query*—.de la siguiente manera. Sea $\mathbf{w}_j$ el vector de pesos de la neurona *j* de la capa competitiva, que se corresponde con el vecino *j* del nodo. Sea h_j el número de *QueryHits* devueltos para esa búsqueda desde el vecino *j*.

$$\mathbf{w}_j(t+1) = \begin{cases} \mathbf{w}_j(t) + h_j\alpha[\mathbf{v} - \mathbf{w}_j(t)], & \text{si } h_j \geq 1 \\ \mathbf{w}_j(t) - \alpha[\mathbf{v} - \mathbf{w}_j(t)], & \text{si } h_j = 0 \end{cases} \quad (5)$$

En caso de acierto, la adecuación del peso será proporcional al número de *QueryHits* devueltos por el nodo vecino. Además se ha utilizado una velocidad de aprendizaje constante —α no depende del tiempo—. Ello se debe a que, asumiendo una tasa alta de cambios en la red P2P, no se espera que el entrenamiento LVQ vaya convergiendo a una clasificación estable. Sólo se utiliza para atenuar el impacto de los cambios producidos para poder retrasar la propagación de mensajes *NeuralInf* que ha demostrado en la experimentación ser un método más eficiente.

Difusión de Información de Actualización combinada con LVQ1. El entrenamiento de las redes neuronales por medio del algoritmo LVQ1 expuesto ha demostrado converger de manera lenta. La naturaleza generalmente cambiante de los sistemas P2P hace aconsejable la propagación de mensajes *NeuralInf* para actualizar rápidamente los pesos de las neuronas en los nodos de la red. Sin embargo, aunque el rendimiento del entrenamiento neuronal es inferior en cuanto a la velocidad de adecuación, aventaja a la difusión de información de actualización en el hecho que no genera tráfico alguno en la red. Por lo tanto, utilizar ambos en forma complementaria permite reducir el tráfico manteniendo niveles aceptables de rendimiento.

Así es posible espaciar la propagación de mensajes *NeuralInf* realizando entrenamiento neuronal sostenido en todas las intervenciones de búsqueda en la que participen los nodos.

4.4 Estrategia para el Ingreso de un Nodo a la Red P2P.

Cuando un nodo *n* desea ser aceptado como vecino de otro nodo *m*, le envía un mensaje de petición de aceptación. Si *m* acepta, envía a *n* con mensaje *NeuralInf*

especial —TTL=1 para que no sea propagado—, con el vector centroide de actualización calculado para *n* (ver fórmula 3). Una vez que *n* haya sido aceptado por un número mínimo requerido de vecinos —dependiente de las necesidades del nodo—, *n* lanza una difusión ordinaria de mensajes *NeuralInf* quedando concluido el protocolo de ingreso requerido.

5 Experimentación y Resultados Obtenidos

Se procedió a la simulación generando redes P2P con 3000 nodos conectados aleatoriamente con una cantidad de vecinos elegida también de forma aleatoria, $4 \leq cantVecino \leq 12$.

Los casos de prueba fueron diseñados para mostrar cómo afectan al rendimiento del sistema de búsqueda BNAP2P las distintas estrategias de adecuación de pesos neuronales presentadas, y el costo de ancho de banda introducido por ellas. La referencia al algoritmo de búsqueda BFS de *Gnutella* en la tabla 1 es sólo ilustrativa.

Tabla 1. Comparativa entre los métodos de adaptación propuestos. En la primera columna DLP_i hace referencia al método DLP con TTL igual a *i*, y $DLPA_{(i,j)}$ hace referencia al método DLPA con TTL igual a *i* y porcentaje de selección de vecinos igual a *j*.

Método Búsqueda / Actualización	Mensajes Query enviados	Porcentaje de recursos hallados	Mensajes NeuralInf enviados
BFS de Gnutella	4372.65	72.80 %	
BNAP2P/Naive	1975.95	86.83 %	21273.00
BNAP2P/DLP_3	1971.68	85.71 %	618.65
BNAP2P/$DLPA_{(3\ ,\ 75)}$	1951.50	84.29 %	438.47
BNAP2P/DLP_1	2077.30	79.86 %	8.75

Para medir los efectos del enfoque híbrido presentado en la sección 4.3 se operó de la siguiente manera: Luego de una adecuación Naive de todos los nodos de la red se procedió a la simulación de 1000 operaciones de búsqueda —con y sin entrenamiento LVQ1— simulando 1 cambio en el conjunto de recursos compartidos cada 2 operaciones de búsqueda. Luego se midió la forma en que estos cambios degradaron el rendimiento del sistema de búsqueda (Ver tabla 2).

Tabla 2. Comparativa de BNAP2P con y sin entrenamiento LVQ1.

	Mensajes Query enviados	Porcentaje de recursos hallados
Antes de simular los cambios	1975.95	86.83 %
BNAP2P sin LVQ1	1976.02	84.17 %
BNAP2P con LVQ1 (α=0.05)	1976.01	84.91 %

La Tabla 2 muestra que el entrenamiento LVQ1 realizado mejora levemente los resultados. Lo destacable del caso es que no requiere ancho de banda adicional.

En cuanto a la estrategia RSC presentada, aún no se han obtenido resultados experimentales concluyentes.

6 Conclusiones

Se ha presentado un conjunto de estrategias de adaptación de las redes neuronales para hacer frente a cambios en el ambiente para el método de Búsqueda Neuronal de Recursos. Las pruebas realizadas muestran que para reducir el requerimiento de ancho de banda del subsistema de adaptación, con menor pérdida de rendimiento, es conveniente utilizar el enfoque híbrido junto a una estrategia de propagación DPL con un valor pequeño de TTL. Los resultados obtenidos en la evaluación de la estrategia DPLA aún no resultan concluyentes.

Referencias

1. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks", in Proceedings of the 16th International Conference on Supercomputing, ACM Press, pp.84-95. (2002)
2. V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yatzi, "A local search mechanism for peer-to-peer networks", in Proceedings of the 11th International Conference on Information and Knowledge Management, ACM Press, pp. 300-307. (2002)
3. B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02), (2002).
4. D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks", in Proceedings of the Third IEEE International Conference on P2P Computing (P2P2003), IEEE Press, pp. 102-109. (2003)
5. Vapa M., Kotilainen N., Auvinen A., Töyrylä J.,Hyytiälä H., Vuori J., "NeuroSearch: evolutionary neural network resource discovery algorithm for peer-to-peer networks", University of Jyväskylä, (2004)
6. Ardenghi Jorge y Echaiz Javier. "Peer-to-peer systems: the present and the future". Journal of Computer Science & Technology. JC&T Vol.7 No.3 October (2007)
7. Yatin Chawthe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, Scott Shenker. "Making Gnutella-like P2P systems scalable". SIGCOMM'03, August 25-29, 2003, Karlsruhe, Germany. Copyright 2003 ACM 1-58133-735-4/03/0008
8. Corbalan Leonardo, Lanzarini Laura y De Giusti Armando. "Resources NeuroSearch in Peer-to-Peer Networks". 31st International Conference on Information Technology Interfaces. University of Zagreb, University Computing Centre. Cavtat/Dubrovnik, Croatia. June 22-25, pp 597 – 602 (2009)
9. Corbalan Leonardo, Lanzarini Laura y De Giusti Armando. "Búsqueda Neuronal de Recursos con Exploración Incremental en Redes Peer-to-Peer". Jornadas Chilenas de Computación. JCC 2009. Santiago de Chile. Chile. 9 al 14 de noviembre de 2009. XIII Workshop on Distributed Systems and Parallelism (WSDP). pp 11-20. (2009)